Software per la simulazione Monte Carlo e la ricostruzione di vertici primari

Enrica Bergalla[1], Valerio Pagliarino[2]

[1] Studentessa C.d.L. Magistrale in Fisica, Università degli Studi di Torino - enrica.bergalla@edu.unito.it
[2] Studente C.d.L. Magistrale in Fisica, Università degli Studi di Torino - valerio.pagliarino@edu.unito.it

[1,2] Gli autori hanno contribuito ugualmente

I. Specifiche tecniche e descrizione dell'esperimento

In questa breve relazione viene presentato un software per la simulazione Monte Carlo e la ricostruzione di vertici primari di tipo *fast* in un esperimento caratterizzato da:

- Due fasci di particelle ad alta energia che collidono in un *collision diamond* con distribuzioni delle coordinate del vertice gaussiane.
- Un *beam-pipe* in berillio del diametro di 30 mm e dello spessore di 0.8 mm
- Due strati di rivelatori a pixel al silicio "ideali" con diametri medi di 40 mm e 70 mm, spessore di 0.2 mm e lunghezza di 270 mm.

Vale l'ipotesi di particelle ad altissimo momento, di conseguenza si trascurano gli effetti del campo magnetico.

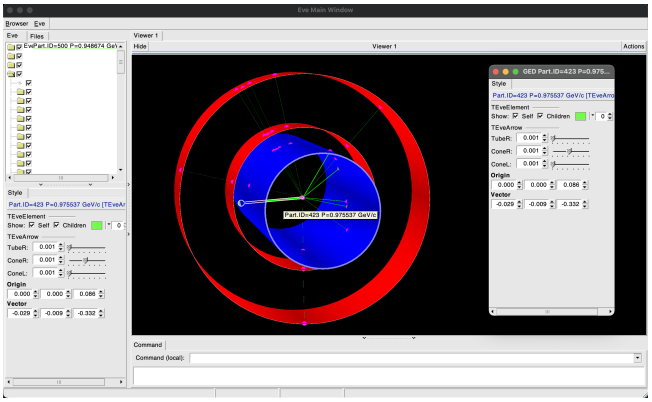

Figura 1. Geometria dell'esperimento sovrapposta ad alcune tracce di particelle, rappresentata nell'event display.

Il software si occuperà di simulare eventi caratterizzati da un vertice primario per evento che produce una determinata molteplicità di particelle (la distribuzione di molteplicità è fornita dall'utente). Le particelle interagiscono con i materiali e i rivelatori unicamente per scattering multiplo (gli altri effetti vengono trascurati) e provocano l'accensione di opportuni pixel nei rivelatori. Il software di simulazione registra le coordinate dei punti (*hits)* di interazione particelle-rivelatore, le coordinate dei vertici primari per ciascun evento.

L'algoritmo di ricostruzione, invece, legge i dati descritti e a partire dai punti di intersezione particella-rivelatore e stima le coordinate dei vertici primari. Sull'output di tale algoritmo viene successivamente eseguita un'analisi che include:

- Valutazione dell'efficienza al variare della molteplicità
- Valutazione dell'efficienza al variare della posizione del vertice primario nella *verità Monte Carlo*
- Distribuzione dei residui, per diversi intervalli di molteplicità
- Valutazione della risoluzione al variare della molteplicità
- Valutazione della risoluzione al variare della posizione del vertice primario nella *verità Monte Carlo*

II. Caratteristiche della simulazione Monte Carlo

Il presente software di simulazione Monte Carlo, utilizza un approccio molto semplificato al problema della simulazione del trasporto di particelle attraverso la materia. Dal momento che tutti i materiali con cui la particella interagisce sono considerati spessori sottili e hanno forme geometriche che è immediato descrivere come superfici parametriche, non vengono utilizzati metodi numerici con discretizzazione della traiettoria della particella (come avviene, ad esempio, in *Geant 4*), ma anche le traiettorie vengono trattate come curve parametriche e si effettua il calcolo tramite formule, in maniera analitica. Questo rende preferibile utilizzare come oggetto di base della simulazione non una classe corrispondente alla singola particella, ma alla singola traccia; quindi, ad esempio, un'interazione di scattering multiplo, che non distrugge la particella, provoca la distruzione di un oggetto traccia e la costruzione di uno nuovo.

Il software di simulazione include la cinematica relativistica, quindi le particelle vengono generate dal vertice primario con massa a risposo m_0 e opportune distribuzioni del momento $|\vec{p}|$ fornite dall'utente, si propagano con velocità $\vec{v} = \vec{p}/\gamma m_0$ e interagiscono con i materiali ad istanti di tempo t. Dal momento che il sistema di equazioni per ottenere l'intersezione traccia-geometria tratta le traiettorie come rette e restituisce quindi più soluzioni, viene proprio utilizzato il confronto tra i tempi di impatto per stabilire quale tra le possibili *hit* si verifica realmente. Di contro, tenere conto della cinematica relativistica implica il calcolo di numerosi fattori γ ed è ben noto che l'operazione di radice quadrata richiede un tempo macchina considerevole, questo penalizza le prestazioni del presente software.

III. Modello di trasporto e simulazione dell'interazione radiazione-materia

Il modello di trasporto di particelle cariche implementato in questo software è estremamente semplice e al momento prevede esclusivamente la simulazione dello scattering multiplo coulombiano. Il fenomeno è trattato nell'approssimazione di spessori sottili, quindi con traslazione della traccia nell'attraversamento del materiale nulla. $\Delta\vec{x} = 0$.

La distribuzione dell'angolo di deflessione che viene campionata con la tecnica Monte Carlo è gaussiana, con valore medio $\theta_0 = 0$ e deviazione standard θ_{rms}. Quest'ultima è data da $\theta_{rms} = \sqrt{2}\theta_{rms,plane}$ dove $\theta_{rms,plane}$ viene ottenuto utilizzando la formula di Highland [1]:

$$\theta_{rms,plane} = \frac{13.6MeV/c}{p\beta} z \sqrt{\frac{x}{X_0}}(1 + 0.038ln(x/X_0)) \quad (1)$$

Nel caso specifico della nostra simulazione, considerando particelle ad alto momento, è possibile un'ulteriore approssimazione di ordine zero in cui si pone θ_{rms} = 1 mrad. Nel programma questa approssimazione può essere abilitata tramite un parametro contenuto nello `struct` denominato *physicsList*, in caso contrario il valore della lunghezza di radiazione X_0, di z e lo spessore del materiale vengono letti dal file di configurazione. Il valore dell'impulso della particella e il rispettivo β sono invece contenuti nella proprietà della traccia in corso di elaborazione.

IV. Algoritmo di ricostruzione dei vertici

La simulazione implementata è di tipo *fast 2*, quindi l'accensione dei singoli pixel e la risposta del rivelatore non viene simulata, ma semplicemente se ne tiene conto con un processo di smearing gaussiano.
Dunque, a partire dalle *hit*, vengono ricostruite le tracce delle particelle e la posizione del vertice ricostruito è data dall'intersezione di queste tracce con l'asse z del sistema (come asse z si considera l'asse del fascio).
E' possibile valutare la performance del software confrontando le coordinate z ottenute in uscita dalla ricostruzione con le coordinate z del vertice primario (Verità Monte Carlo). I dettagli della ricostruzione saranno approfonditi nella sezione B del prossimo paragrafo.

V. Architettura del software

Il software è implementato in *plain C++17* con l'utilizzo del framework *CERN ROOT* e ha un'architettura *object-oriented*. Il programma non prevede un *makefile*, ma uno script di compilazione che viene eseguito dall'interprete *Cling* di *ROOT* ed esegue la compilazione del restante codice sorgente utilizzando *ACLiC*. [3]

Il software prevede alcuni script interpretati da *Cling* dotati di interfaccia grafica, attraverso i quali l'utente può generare i files di configurazione in formato ASCII e i files in formato *ROOT* contenenti le distribuzioni iniziali da utilizzare per la simulazione delle collisioni. E' quindi possibile avviare il programma dalla linea di comando, specificando con un argomento se si vuole utilizzare l'algoritmo di simulazione oppure di ricostruzione (*sim* / *rec*) e passare come parametro il percorso relativo al file di configurazione. All'interno di tale file saranno specificati anche i percorsi al file *ROOT* di input e al file *ROOT* di output.

```
root -l -b 'start.cxx("sim",
 "./simulationConfig.txt")'
```

allo stesso modo viene eseguito il programma di ricostruzione

```
root -l -b 'start.cxx("rec",
 "./reconstructionConfig.txt")'
```

Questo approccio permette di eseguire i codici di simulazione e ricostruzione senza l'intervento interattivo dell'utente e questo rende più semplice l'eventuale integrazione con un *job scheduler* o l'esecuzione di più processi in parallelo.

Altrimenti è possibile avviare la simulazione in modalità interattiva avviando la macro interpretata *start.cxx* e successivamente chiamare la funzione

```
Cli::Start("sim",
 "./simulationConfig.txt")
```

Come parametro aggiuntivo, oltre a *sim* è possibile inserire la keyword *persist* per eseguire la simulaione in modalità con persistenza di tutte le tracce, sovrascrivendo la relativa opzione indicata nel file di configutazione; tale modalità verrà descritta più in dettaglio in seguito.

Nell'organizzazione delle classi è stata scelta un'architettura che ottimizzi la semplicità di estensione e di aggiunta di nuove funzionalità, al costo di aumentare leggermente il livello di astrazione e di conseguenza la lunghezza del codice sorgente.

A. *Simulazione*

Per avviare una simulazione viene creato un nuovo oggetto della classe *RunManager*, una classe container che eredita da *TTree* contenente due *TBranch*: il primo registra i vertici primari, mentre il secondo i punti di interazione tra le particelle e i due piani di rivelatori al silicio. La classe *RunManager* contiene, inoltre, un vettore di puntatori a tutti gli oggetti che rappresentano gli eventi relativi a quella run. Si assume per semplicità che il *trigger* dell'esperimento generi eventi con sempre un solo vertice primario da ricostruire, con assenza di eventi vuoti e di *pile up*. A questo punto, viene istanziato un oggetto della classe *ProgramConfig* che leggerà il file di configurazione ed effettuerà la lettura del *TFile* contenente i dati di input; successivamente il costruttore di default di *RunManager* esegue in sequenza le seguenti operazioni:

- Viene istanziato un oggetto della classe *RndEngine* che eredita da *TRandom3* ed è (l'unico) generatore di sequenze di numeri pseudo-casuali utilizzato dall'intero software di simulazione. Il *seed* viene impostato in questa fase. Il puntatore a questo oggetto verrà fornito alle classi che implementano i metodi Monte Carlo. Tale *PRNG* utilizza l'algoritmo Marsenne-Twister per la generazione delle sequenze. [2]
- Viene istanziato un oggetto della classe *ExperimentSimulation*, il quale contiene la descrizione della geometria dell'esperimento e si occuperà di chiamare i metodi statici contenuti nella classe *TransportEngine* per simulare

l'interazione radiazione-materia. A tale oggetto vengono passati i puntatori a *ProgramConfig* configurazione e all'oggetto PRNG. Viene a questo punto chiamato il metodo *BuildGeometry()* che inizializzerà la geometria.
- Viene istanziato un oggetto della classe *ParticleGun* che si occupa di simulare le collisioni, la generazione dei vertici primari e la generazione delle tracce delle particelle uscenti dai vertici primari.
- Viene, infine, inizializzato un oggetto della classe *DetectorEffects* che contiene i metodi per la simulazione delle *soft particles* e di altri effetti che producono l'accensione di pixel del rivelatore al di fuori della traiettoria di particelle provenienti dai vertici primari.

A questo punto tramite il metodo *RunManager::BeamOn()* viene avviata la simulazione, la quale è gestita a più basso livello dal metodo *RunManager::SimulationBackend()* il quale esegue in sequenza le seguenti operazioni, ripetendole per il numero di eventi prescelto nel file di configurazione:

- Viene generato il nuovo evento, allocandolo sullo *heap*, dal momento che la classe *EventManager* eredita da *TObject* e supporta la persistenza su disco, il buffer verrà periodicamente svuotato scrivendo nel file ROOT di output.
- Viene generato un determinato numero di collisioni per evento: per ciascuna collisione viene invocato il metodo *GenerateCollision()* della classe *ParticleGun* che genera un certo numero di tracce uscenti dal singolo vertice primario, in base alla distribuzione di molteplicità presente nel file *ROOT* di input.
- L'evento viene passato al metodo *ProcessEvent()* della classe *ExperimentSimulation*, la quale si occuperà di simulare l'interazione di ciasuna traccia proveniente dal vertice primario con il primo oggetto che interseca in ordine di tempo, a questo punto la traccia verrà segnata come *non attiva* e una nuova traccia *attiva* verrà generata e seguita fino al suo primo punto di interazione. Il ciclo si arresta quando tutte le tracce hanno interagito con l'ultimo materiale sensibile (piano esterno di rivelatori al silicio). Le tracce sono oggetti di una classe che eredita da *TObject*, sono allocati sullo *heap* e ogni evento contiene un vettore di puntatori a tutte le tracce che gli appartengono. Le tracce, le hit e gli eventi sono rappresentati da classi che supportano l'archiviazione persistente di ROOT e nel momento in cui vengono generati la *directory* di *default* è il file ROOT di output, questo anche ai fini di ottimizzare l'utilizzo della RAM.
- Vengono infine simulati gli effetti di rumore con l'accesione di pixels in base a distribuzioni assegnate nel file di *ROOT* in ingresso, in accordo con quanto specificato nel file di configurazione. Gli istogrammi nel file ROOT in ingresso specificano sia la distribuzione spaziale (*TH2D*) consentendo, ad esempio, di simulare rumore non uniformemente distribuito perché alcune parti del rivelatore possono aver subito danno da radiazione maggiore o sono più soggette a interazione con *soft particles*, sia la distribuzione (gaussiana o poissoniana) del numero di conteggi di rumore (*TH1D*).

Una *run* può essere processata con l'impostazione *booleana* di configurazione *SingleEventPersistenceEnabled* allo stato vero oppure falso. Nel primo caso, tutti gli eventi e le tracce generate non verranno mai deallocate e il buffer verrà periodicamente svuotato nel file *ROOT* di output. Nel secondo caso, invece, al termine dell'evento esso viene deallocato e di conseguenza tutte le sue tracce vengono eliminate dalla memoria. Gli eventi contengono inoltre un vettore di puntatori ad oggetti della classe *Hit* che descrivono le interazioni di ogni traccia con i rivelatori o altri materiali, anche essi vengono deallocati nel secondo caso.

Rimangono invece sempre memorizzati nei *TBranch* dell'oggetto della classe *RunManager* (che eredita da *TTree*), le posizioni dei vertici primari e le interazioni con il rivelatore, associate al vertice da cui proveniva la particella che le ha generate. In questo modo, in fase di analisi, sarà disponibile una "verità Monte Carlo" sulla base della quale valutare la bontà dell'algoritmo di ricostruzione.

La modalità con persistenza su disco di ogni singola traccia, sebbene molto più dispendiosa dal punto di vista della memoria e delle operazioni di I/O, può essere utile se consideriamo che il software mette a disposizione un *event display*, tramite l'omonima classe, che permette di visualizzare al termine della simulazione un determinato evento renderizzando tracce e geometrie in 3D tramite il motore *OpenGL*. La modalità con persistenza come *default* risulta disabilitata.

La fisica, per quanto riguarda la simulazione, risulta quindi implementata quasi interamente nella classe con metodi statici *TransportEngine* e in alcuni metodi *getter* e *setter* della classe *Track* che calcolano grandezze tipiche della cinematica relativistica a partire dall'impulso e viceversa.

Il software prevede supporto per particelle di varia massa e carica elettrica, ma la funzionalità non è al momento utilizzata. Non è previsto al momento il supporto a campi magnetici, ma la struttura modulare delle classi e la gestione della cinematica rende possibile una sua eventuale aggiunta in futuro.

B. *Ricostruzione*

L'intero processo di ricostruzione e analisi è gestito dalla classe *VertexReco* che crea delle istanze e utilizza i metodi delle classi *CalculateDeltaPhiMax*, *HitsAnalysis* e *ResultsAnalysis*. In particolare:

- La classe *CalculateDeltaPhiMax* utilizza l'informazione di verità Monte Carlo che associa ad ogni *hit* il numero della particella che l'ha generata. In questo modo, è possibile costruire una distribuzione della differenza di angolo azimutale tra due *hit*, sui due diversi rivelatori, associate alla medesima particella. Questa classe è totalmente indipendente dalla ricostruzione vera e propria, infatti la verità Monte Carlo non è nota in fase di ricostruzione e, in questo caso, viene solamente utilizzata per avere una stima plausibile del valore di differenza di angolo azimutale massima ($\Delta\phi_{max}$) associata alle *hit* sui due rivelatori della stessa particella. E' comunque possibile per l'utente disabilitare questa funzionalità: in questo caso, $\Delta\phi_{max}$ viene impostato al valore di 20 mrad.
- Nella classe *HitsAnalysis* sono implementati i metodi per analizzare le *hit* prodotte dalla simulazione. Per ogni

evento, se due *hit* sui due rivelatori hanno una differenza di angolo azimutale inferiore di $\Delta\phi_{max}$, si costruisce la traccia rettilinea passante per questi due punti di impatto (*tracklet*) e, per ognuna, si stima una possibile coordinata z del vertice ricostruito (candidato vertice). La stima vera e propria della z del vertice ricostruito si ottiene tramite il metodo della *Running Window*, di cui l'utente può impostare dimensione della finestra e dimensione dello step. Inoltre, è impostabile anche il numero minimo di candidati vertici che deve contenere una finestra, affinché il vertice in essa ricostruito risulti valido. Allo stesso modo, si imposta anche il maggior numero accettabile di candidati vertici nelle finestre esterne a quella che contiene il massimo numero di candidati vertici. In questo modo si può avere un diverso livello di confidenza sul vertice ricostruito e modificare il trade off tra efficienza e risoluzione variando i parametri impostati. Per ciascun evento, dunque, viene riempito un *TTree*, con un solo *branch*, che contiene la coordinata z del vertice ricostruito e il numero dell'evento corrispondente, il *TTree* viene poi salvato in un file *ROOT* di output della ricostruzione.

- Nella classe *ResultsAnalysis* sono implementati i metodi per analizzare i risultati della ricostruzione e confrontarli con la Verità Monte Carlo: in particolare, si vogliono valutare l'efficienza e la risoluzione con cui si ricostruisce il vertice in funzione della z del vertice primario e della molteplicità. A tale scopo, vengono utilizzate due serie da due istogrammi bidimensionali (*TH2D*). Per quanto riguarda lo studio dell'efficienza, si creano due istogrammi 2D identici aventi sull'asse x e y la z del vertice primario (Z_{true}) e la molteplicità. Se, per un determinato evento, il vertice viene ricostruito, si riempiono entrambi gli istogrammi, in caso contrario viene riempito solo il primo dei due. Poiché si vogliono considerare determinati intervalli di Z_{true} e molteplicità in cui valutare l'efficienza, usando $TProfile$, è possibile prendere in considerazione specifici range della variabile di interesse e proiettare la corrispondente sezione dell'istogramma 2D sull'asse opposto rispetto alla variabile di interesse, ottenendo così degli istogrammi 1D che sono le distribuzioni marginali di probabilità. Per ogni range di interesse, l'efficienza è ottenuta come il rapporto tra il numero di ingressi dell'istogramma 1D ottenuto dal secondo $TH2D$ e il numero di ingressi dell'istogramma 1D ottenuto dal primo $TH2D$. L'incertezza sull'efficienza è considerata di tipo binomiale.
  Per quanto riguarda lo studio sulla risoluzione, si costruiscono due istogrammi bidimensionali, il primo avente sull'asse x e y, rispettivamente, i residui e la molteplicità ed il secondo i residui e la Z_{true}. Per ciascun evento in cui la ricostruzione va a buon fine, vengono riempiti i due istogrammi. Dunque, analogamente a quanto spiegato in precedenza, dal primo $TH2D$ vengono ottenuti dei $TH1D$ nei diversi intervalli di interesse per la molteplicità. Questi istogrammi 1D vengono fittati con una distribuzione gaussiana e si considera come risoluzione la σ del fit con relativo errore associato. In via del tutto analoga è possibile ripetere la stessa operazione considerando il secondo $TH2D$ e gli intervalli di interesse in Z_{true} per valutare la risoluzione in funzione della coordinata del vertice primario.
  Tutti i grafici in output dell'analisi e gli istogrammi dei residui per determinati range di molteplicità sono salvati in un file $ROOT$ e visualizzabili dall'utente mediante un $TBrowser$.

C. *Event display*

Il presente software di simulazione prevede un *Event Display* grafico basato sul motore di rendering 3D *OpenGL* per la visualizzazione interattiva della geometria e delle tracce di un singolo evento, realizzato utilizzando le classi del *Event Visualization Environment* di *ROOT*.

La classe *EventDisplay* è in grado di importare dalla classe *ExperimentSimulation* la geometria dell'esperimento in modo automatico, generando le opportune classi renderizzabili *TEveGeoShape*. A questo punto è possibile caricare un evento della classe *EventManager*; il metodo *LoadEvent* si occuperà di disegnare tutte le tracce presenti.

Il modo più semplice per utilizzare l'*event display* è simulare una run con persistenza delle singole tracce (nell'ordine di 10^2 eventi) in modalità interattiva, caricando la macro (interpretata) *start.cxx* e digitando successivamente il comando

```
Cli::Start("sim persist",
 "./simulationConfig.txt")
```

A questo punto, è possibile visualizzare un evento utilizzando passando l'eventID (che parte da 1) alla funzione apposita come nel comando seguente:

```
Cli::DrawEvent(4)
```

Si otterrà una finestra simile a quella illustrata di seguito.

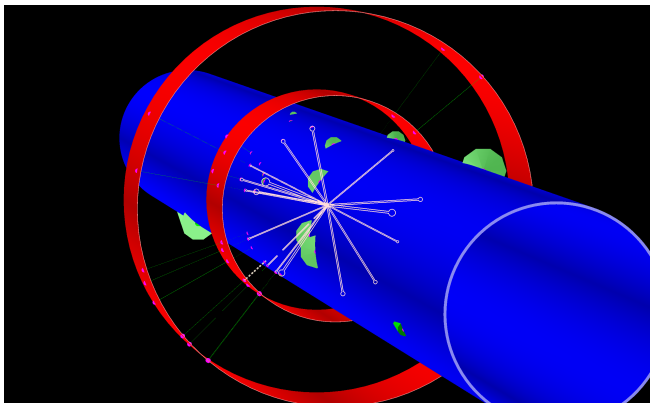

Figura 2. Un evento visualizzato nell'event display

VI. Interfaccia utente

L'interfaccia utente del software si compone due macro ROOT interpretabili con *Cling*:

- **simulationConfig.cxx** - Macro con interfaccia utente grafica (GUI) che permette di produrre il file di configurazione e il file *ROOT* di input per eseguire la simulazione di una run.

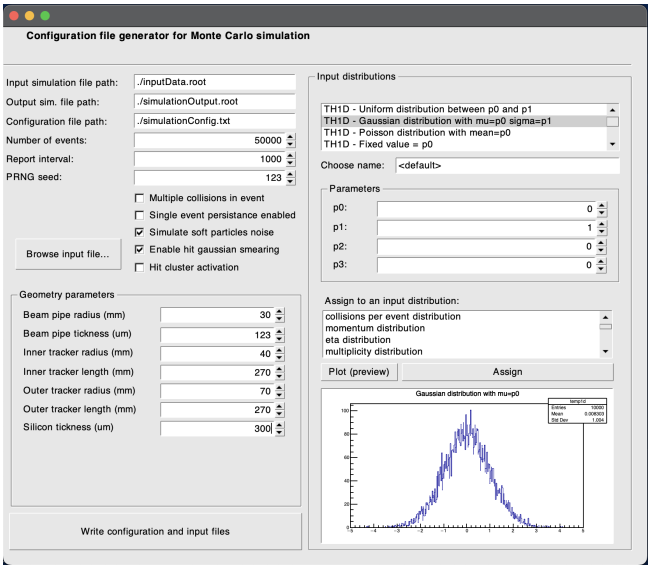

Figura 3. Interfaccia grafica del generatore di files di configurazione per la simulazione

- **reconstructionConf.cxx** - Macro con interfaccia utente grafica (GUI) che permette di produrre il file di configurazione per eseguire la ricostruzione di una determinata run, a partire dal file ROOT prodotto dalla simulazione.

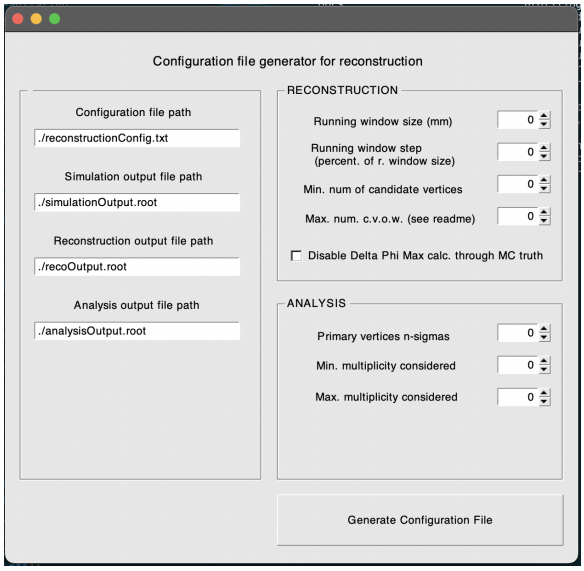

Figura 4. Interfaccia grafica del generatore di files di configurazione per la ricostruzione

L'interfaccia grafica per generare il file di configurazione per la simulazione permette, oltre che di inserire valori numerici, anche di generare le distribuzioni di apposite variabili fisiche quali la molteplicità di tracce per evento, la pseudorapidità eccetera. Una volta digitato il percorso al file ROOT di input che conterrà tali distribuzioni nelle caselle di testo sulla sinistra, sarà possibile scegliere una funzione densità di probabilità tra un elenco di opzioni, configurarne i parametri, selezionare da un secondo elenco la variabile fisica a cui assegnarla e infine premere il pulsante *Assign*. Alla pressione del pulsante la macro genererà un apposito istogramma e lo scriverà nel file ROOT di input, dopodiché andrà ad aggiungere l'assegnazione al file di configurazione in formato testo, in modo da informare il software di simulazione sul nome della chiave da utilizzare per la lettura dal *TFile*. Se il pulsante *Assign* viene premuto più volte, rimarrà valida l'ultima assegnazione. E' anche possibile scegliere dal primo elenco un'opzione corrispondente ad un istogramma già presente nel *TFile* in ingresso (in tal caso si fornisca il percorso ad un file ROOT non vuoto), in questo caso è necessario inserire nella casella di testo *Choose name* il nome della chiave che il software di simulazione dovrà utilizzare per accedere a tale oggeto. A questo punto è possibile selezionare la variabile fisica e premere *Assign*. Con questa modalità sono state caricate nel software di simulazione le distribuzioni *heta* e *hmul* fornite dal docente.

E' importante notare che non è obbligatorio configurare una distribuzione per tutte le variabili fisiche: ogni voce presente nel file di configurazione produrrà da parte del programma di simulazione un'operazione di sovrascrittura dei valori numerici o delle distribuzioni di default che vengono inizializzate nel file *conf.cpp* all'avvio della simulazione. In assenza di una specifica voce, vengono utilizzati tali valori di default *hard-coded*.

In alternativa è sempre possibile utilizzare *TBrowser* per accedere ai files ROOT e un qualsiasi editor di testo per modificare i files di configurazione. In tal caso notare che il sistema di unità di misura utilizzato dal programma è il Sistema Internazionale (MKS). Nel corpo del codice sorgente è possibile utilizzare una collezione di costanti moltiplicative globali come *"GeV", "ns",* etc... per inserire i valori numerici nelle unità desiderate.

L'interfaccia grafica per la ricostruzione permette di impostare il path ai file di root in uscita da simulazione, ricostruzione e analisi. Nella sezione della ricostruzione, si possono impostare la lunghezza della $Running\ Window$ e la lunghezza dello step. Inoltre sono impostabili il numero minimo di candidati vertici che una finestra deve contenere affinché il vertice in essa ricostruito venga preso in considerazione e il numero limite di candidati vertici che possono contenere le finestre esterne a quella che contiene il maggior numero di candidati vertici perché la ricostruzione vada a buon fine (questo valore va espresso in percentuale rispetto al numero di candidati vertici massimo contenuto in una finestra).
Infine, nella sezione di analisi, è possibile impostare il numero di σ del vertice primario all'interno delle quali verranno considerati i vertici primari in fase di analisi e la minima e massima molteplicità per cui verranno condotti gli studi di efficienza e risoluzione.
E' possibile non modificare uno o più dei valori presenti nell'interfaccia, in tal caso verrà impostato il default.

VII. SIMULAZIONI DI TEST

Per la simulazione di test si sceglie un numero di eventi pari a 350000 per i quali viene considerata una singola collisione per evento. Le distribuzioni di pseudorapidità e molteplicità e la geometria del sistema sono quelle fornite dal professore,

mentre le coordinate del vertice primario sono distribuite gaussianamente con valore medio corrispondente all'origine degli assi e deviazione standard di 0.1 mm sull'asse x e y e di 5.3 cm sull'asse z. Il numero di hit dovute al rumore su entrambi i rivelatori è generato con distribuzione poissoniana centrata in 6.5 e la loro posizione è distribuita uniformemente su tutta la superficie dei detector.
Per quanto riguarda la ricostruzione, viene utilizzata una dimensione della *Running Window* pari a 5 mm ed uno step di 0.5 mm. Inoltre, viene scelto di ricostruire il vertice solo se nella finestra è presente almeno un candidato vertice e se nelle finestre esterne a quella che massimizza il numero di candidati vertici non ne sono presenti di più che in quest'ultima.
In fase di analisi, sono stati considerati solo eventi in cui il vertice primario è incluso all'interno di 3σ nella sua distribuzione gaussiana e molteplicità comprese tra 3 e 52.

VIII. Analisi e discussione dei risultati

Di seguito sono mostrati i due istogrammi 2D che riportano rispettivamente molteplicità e residui oppure Z_{true} e residui. Si nota che il primo dei due è evidentemente discretizzato, a causa del fatto che i valori di molteplicità sono interi e si nota che il profilo dell'istogramma 2D lungo l'asse delle ascisse riflette la distribuzione di molteplicità.

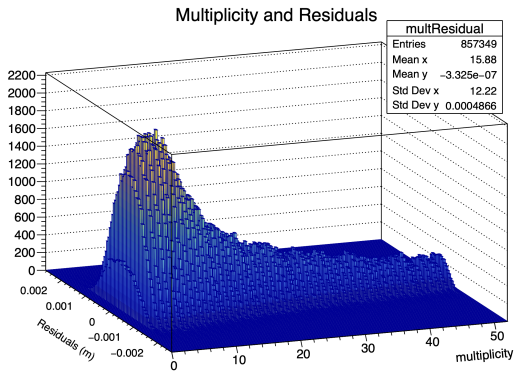

Figura 5. Istogramma bidimensionale di residui e molteplicità

Invece, per il secondo istogramma 2D, l'andamento non è discretizzato perché i valori di Z_{true} sono continui e anche in questo caso il profilo lungo le ascisse riflette la distribuzione della z del vertice primario.

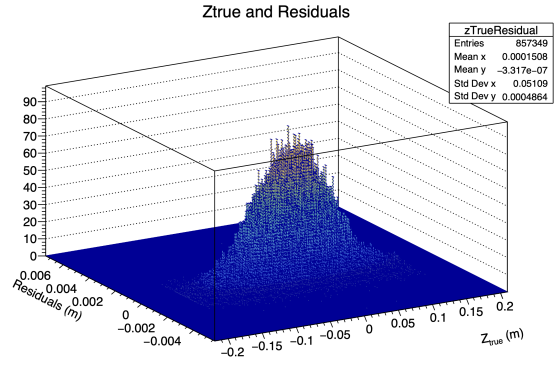

Figura 6. Istogramma bidimensionale di residui e Z_{true}

Di seguito sono riportati l'andamento dell'efficienza in funzione di molteplicità e Z_{true}. Le incertezze associate all'efficienza sono stimate con formula binomiale.
Nel primo caso si nota una saturazione dell'efficienza ad 1 per valori di molteplicità pari o superiori a 20, vale a dire che per alti valori di molteplicità è possibile ricostruire il vertice nella quasi totalità dei casi. Nel secondo caso, invece, la massima efficienza è raggiunta per i valori di Z_{true} più vicini all'origine. In questo caso, si nota che la saturazione non avviene ad un valore di efficienza pari ad 1, bensì, a 0.84 circa. Ciò è dovuto al fatto che, anche per valori di Z_{true} vicini all'origine, sono presenti degli eventi in cui non è possibile ricostruire il vertice perché la molteplicità è troppo bassa.

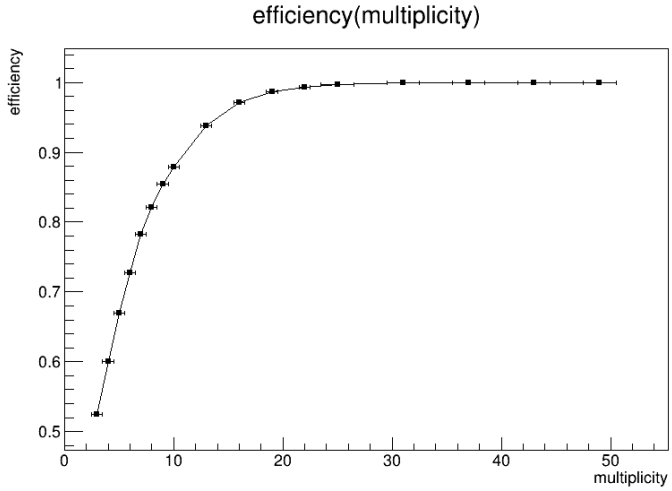

Figura 7. Efficienza in funzione della molteplicità

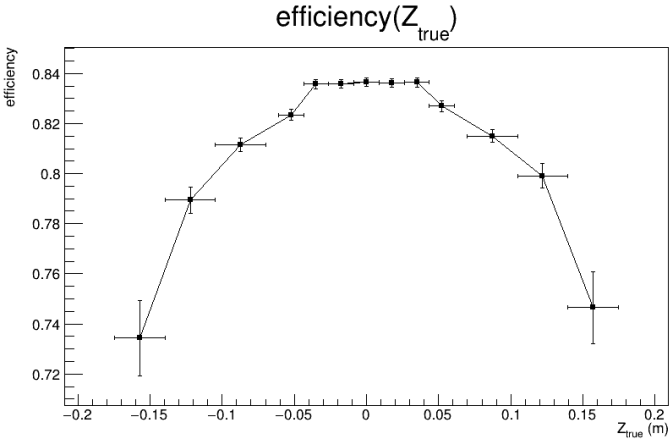

Figura 8. Efficienza in funzione della z del vertice primario

Di seguito, si riportano gli istogrammi dei residui in due specifici range di molteplicità. I dati sono fittati con una distribuzione gaussiana e la risoluzione viene stimata come la deviazione standard calcolata dal fit, a cui è associata come incertezza quella ottenuta dal fit stesso.

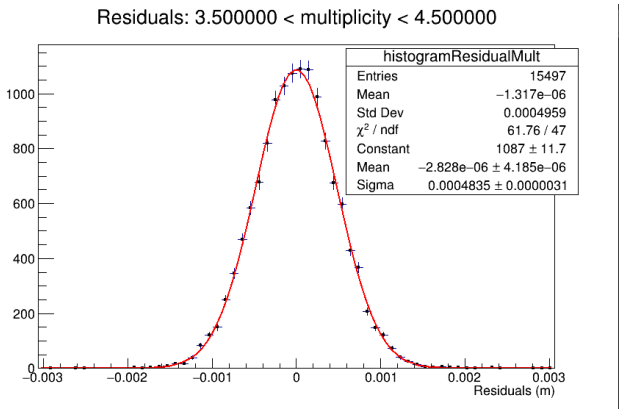

Figura 9. Residui per molteplicità pari a 5

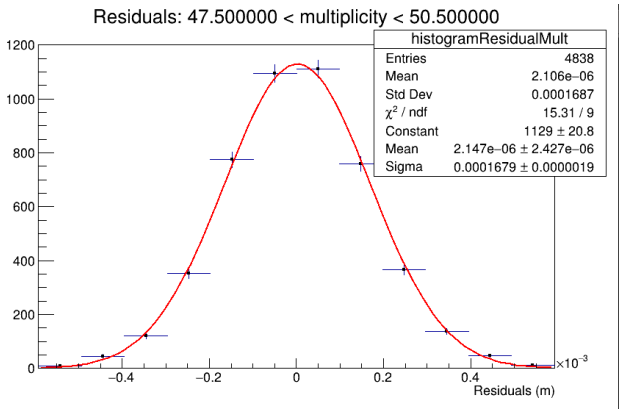

Figura 10. Residui per molteplicità compresa tra 48 e 50

Di seguito sono riportati l'andamento della risoluzione in funzione di molteplicità e Z_{true}.
Si nota come la risoluzione migliori per alti valori di molteplicità a per valori di Z_{true} vicini all'origine.

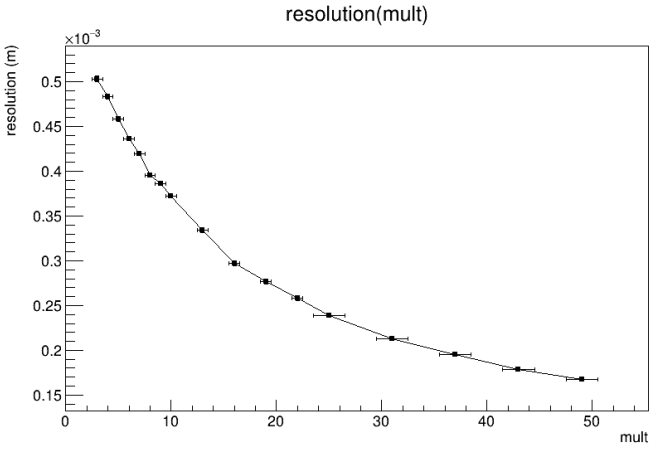

Figura 11. Risoluzione in funzione della molteplicità

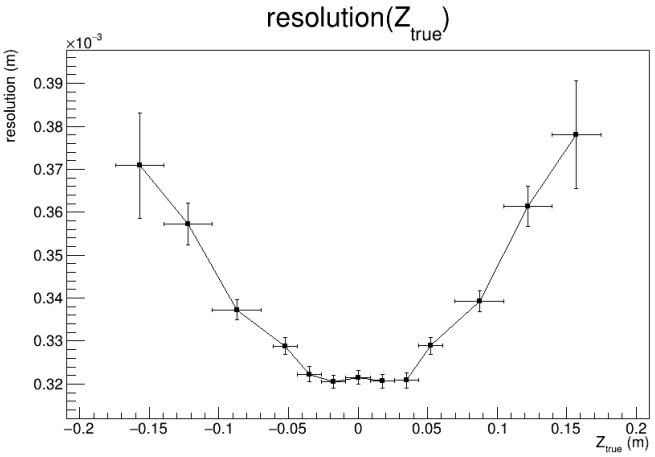

Figura 12. Risoluzione in funzione della z del vertice primario

IX. Benchmark e analisi dell'utilizzo della memoria e della CPU

Le prestazioni del software sono state valutate con diversi benchmark, in particolare è stata eseguita una *run* con 10 milioni di eventi con persistenza delle singole tracce disabilitata, cinematica relativistica abilitata e generazione del rumore abilitata. La run è stata completata in 489.550 s di tempo CPU, corrispondenti a 8 minuti e 10 secondi reali, sulla CPU di un SoC Apple M1 Max, con compilazione nativa ARM 64 bit con 32 GB di RAM.

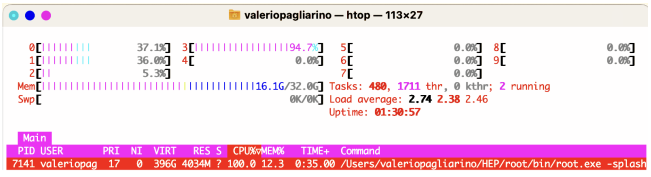

Figura 13. Processo del software di simulazione in esecuzione per la simulazione di 10 milioni di eventi in modalità con oggetti traccia non persistenti

Attraverso un *sampling* dell'utilizzo del tempo CPU da parte del processo è stato possibile ricavare l'albero mostrato in figura 14, dove emerge che l'operazione di scrittura su TFile (in particolare la compressione con l'algoritmo *deflate*) richiede una quantità davvero significativa di tempo macchina, a seguire troviamo la generazione del rumore dovuto alle *soft particles*, con campionamento della distribuzione spaziale presente nel *TH2D* e infine le operazioni trigonometriche inverse necessarie per le trasformazioni di coordinate. Degne di nota sono anche le operazioni di allocazione / deallocazione di memoria attraverso gli operatori *new* e *delete*.

Il grafico dell'allocazione di memoria, nell'esecuzione di 10 milioni di eventi su Apple M1 Max mostra l'andamento rappresentato nella figura seguente, dove i picchi verso il basso sono in corrispondenza delle operazioni di *flush* dei *basket* del *TTree* verso il disco. L'andamento globalmente crescente potrebbe far sospettare la presenza di un *memory leaks*, in realtà se si disabilita lo riempimento dei *TTree* l'allocazione

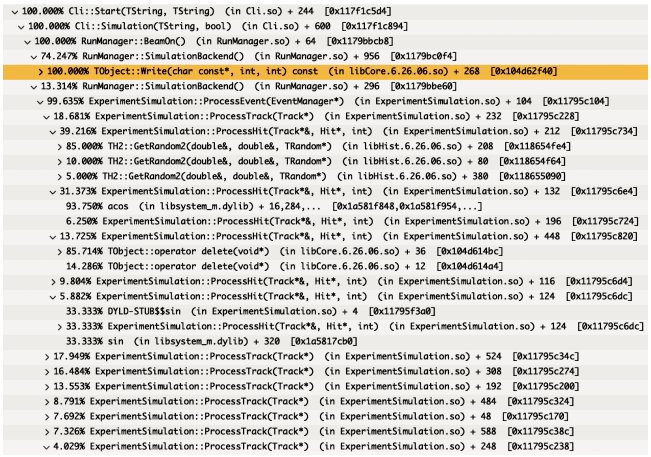

Figura 14. Sampling del tempo CPU, le percentuali mostrano il tempo necessario per l'esecuzione di ciascuna funzione, rispetto al totale del nodo superiore dell'albero

rimane contenuta intorno ai 200 MB e analisi effettuate con tools specifici come *Memleax* su piattaforma Linux Ubuntu AMD 64 [4] non rilevano mancate deallocazioni, ma indicano che nel momento in cui la memoria viene deallocata, essa rimane però frammentata e questo porta gli eventi successivi a richiedere zone di memoria al sistema operativo in un nuovo range di indirizzi. L'utilizzo di RAM mostrato è quello misurato dal comando *top* che comprende anche il *buffer* I/O e la *cache*. *ROOT* per minimizzare il fenomeno della frammentazione di memoria, mette a disposizione la classe *TClonesArray* che prevede il riutilizzo di una determinato range di indirizzi riducendo le chiamate agli operatori *new* e *delete*.

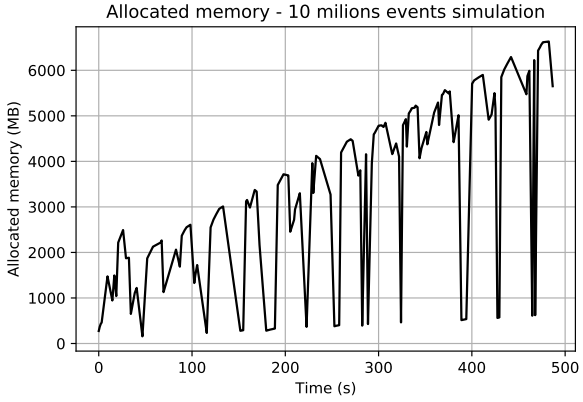

Figura 15. Allocazione di memoria allo scorrere del tempo durante la simulazione di 10 milioni di eventi in modalità con oggetti traccia non persistenti. I picchi corrispondono alle operazioni di svuotamento del buffer verso il disco; l'andamento crescente è dovuto alla frammentazione della memoria rilasciata nella gestione del TTree, siccome non sono stati riscontrati *memory leaks*. La macchina era equipaggiata con 32 GB di RAM.

Il software è scritto in modo tale da scrivere ad intervalli prefissati sul *TFile* un nuovo oggetto *TTree* con stesso nome, ma diverso numero di versione, dopodiché viene invocato il metodo *TTree:Reset()* che provvede a deallocare dalla memoria i dati appena scritti su disco. In fase di lettura, verranno letti sequenzialmente gli eventi di tutte le versioni del *TTree* presenti nel *TFile*. Questo meccanismo permette all'utente, eventualmente, di copiare tra diversi *TFile* blocchi di eventi di dimensioni più piccole più agevolmente; inoltre, consente di garantire l'integrità di una quantità elevata di eventi del *TTree* nel caso in cui il processo di simulazione venga arrestato improvvisamente.

E' invece molto più lenta e richiede naturalmente un maggiore quantitativo di memoria, la modalità con persistenza delle singole tracce necessaria per utilizzare *l'event display*. In tal caso si consiglia di realizzare singole run con un numero di eventi nell'ordine di 10^2.

X. Distribuzione del codice e compatibilità

Il software richiede una versione di *CERN ROOT* uguale o successiva a 6.24/04 (rilasciata il 26/08/2021) poiché il metodo *GetRandom(TRandom *)* della classe *TH1D* che permette il passaggio di una specifico oggetto *TRandom* è stato introdotto a partire da tale *release*[1]. Il software è stato testato in ambiente *Linux Ubuntu 20 LTS* con compilatore *GCC* e in ambiente *Apple MacOS 12.5.1 (Monterey)* (versione di ROOT 6.26/06) con compilatore*Clang-LLVM 13.1.6*. Il software può essere ottenuto clonando il seguente *repository*
https://github.com/valeriopagliarino/TANS.git

XI. Istruzioni per la configurazione, compilazione ed esecuzione del software

Dopo aver ottenuto il codice sorgente e verificato la compatibilità con il sistema operativo e la versione di ROOT installata, è sufficiente decomprimere l'archivio di distribuzione in una cartella in cui il processo di ROOT avrà permessi di lettura e scrittura ed eseguire la macro di compilazione tramite il comando

```
root ./start.cxx
```

Al termine della compilazione, sarà possibile generare un file di configurazione chiamando l'apposita macro

```
root ./simulationConfig.cxx
```

oppure saltare il passaggio se si vuole utilizzare quello di default presente nella medesima cartella in formato *txt*. Altrimenti è possibile modificare manualmente con un qualsiasi editor il file di configurazione fornito, in tal caso prestando attenzione al fatto che le lunghezze sono espresse sempre in metri e il separatore dei decimali è il punto. A questo punto è possibile avviare la simulazione utilizzando i comandi riportati nei paragrafi V e VI.

Analogamente, per generare il file di configurazione di ricostruzione e analisi si può digitare il comando:

```
root ./reconstructionConf.cxx
```

Oppure è possibile editare il file di testo fornito, ricordando gli accorgimenti indicati sopra.

[1]E' possibile utilizzare versioni precedenti cercando nel codice sorgente le chiamate al metodo GetRandom(TRandom *) e rimuovendo l'argomento. In questo modo verrà utilizzato gRandom

XII. Conclusioni

In conclusione, è stato presentato un software di analisi e di ricostruzione vertici di tipo $fast$ 2. Il software presenta un funzionamento corretto ed ha ricostruito in uscita le distribuzioni attese. A livello di performance, il software risulta buono: si riescono a simulare 10 milioni di eventi in circa 10 minuti su un laptop.

Le principali limitazioni risiedono nel fatto che il modello di simulazione sia semplificato: non vengono tenuti in conto eventuali campi magnetici e si considerano solo le particelle di alto momento, per cui le traiettorie possono essere approssimate come rettilinee. Per quanto riguarda lo scattering multiplo, invece, non si è tenuto conto dello spostamento all'interno del materiale, ma solamente della deflessione (approssimazione di materiale sottile) e θ_{rms} è stato assunto pari ad 1 mrad, senza nessuna ulteriore considerazione sul momento della particella. Inoltre, la ricostruzione non supporta ancora eventi con più vertici primari e la X e Y del vertice non vengono ricostruite. La gestione della memoria potrebbe essere notevolmente ottimizzata riducendo il numero di operazioni di new / delete, riutilizzando segmenti dello heap preallocati. In questo modo si ridurrebbe anche la frammentazione.
In generale, si considera il seguente software allineato con le specifiche tecniche prefissate.

XIII. Riferimenti

Riferimenti bibliografici

[1] Gerald R. Lynch e Orin I. Dahl. "Approximations to multiple Coulomb scattering". In: *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 58.1 (mag. 1991), pp. 6–10. DOI: 10.1016/0168-583x(91)95671-y. URL: https://doi.org/10.1016/0168-583x(91)95671-y.

[2] Makoto Matsumoto e Takuji Nishimura. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". In: *ACM Trans. Model. Comput. Simul.* 8.1 (gen. 1998), pp. 3–30. ISSN: 1049-3301. DOI: 10.1145/272991.272995. URL: https://doi.org/10.1145/272991.272995.

[3] Fons Rademakers et al. *root-project/root: v6.20/06*. 2020. DOI: 10.5281/ZENODO.3895852. URL: https://zenodo.org/record/3895852.

[4] *Memleax - debugs memory leak of a running process*. URL: https://manpages.ubuntu.com/manpages/bionic/man1/memleax.1.html.